سید محمد طاها طباطبایی – ۹۸۱۲۷۶۲۸۳۸

گزارش تمرین ۱

چکیده:

در این تمرین، به بررسی ۳ عملیات پایه روی تصاویر می پردازیم. در بخش اول، با گسسته سازی مقادیر پیکسل های تصویر و نحوه تاثیر آن بر کیفیت عکس آشنا می‌شویم. در بخش دوم، دو متد کاهش ابعاد تصویر، یکی حذف سطر و ستون و دیگری میانگین گیری را بررسی می‌کنیم. همچنین برای افزایش ابعاد تصویر نیز، دو متد کپی و تکرار و تبدیل دو خطی را پیاده‌سازی می‌کنیم. در بخش آخر نیز، به بررسی تاثیر تعداد بیت های مورد استفاده در ذخیره سازی تصویر، بر روی کیفیت عکس می‌پردازیم.

۱.۱.۱

توضیح فنی:

در سلول اول، ابتدا به محاسبه هیستوگرام محلی برای تصویر elaine پرداخته ایم.

```
elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)

# equalize elaine image
elaine_histo = calc_hitogram(elaine)
normal_elaine_histo = normalizeHistogram(elaine_histo,elaine.shape[0],elaine.shape[1])
elaine_cdf = calc_cdf(normal_elaine_histo)
equal_elaine = reMap(elaine,elaine_cdf)
```

در اینجا، ابتدا با تابع calc_histogram ، هیستوگرام تصویر را محاسبه می کنیم. سپس با تابع normalizationhistogram ، تصویر را نرمال‌سازی می کنیم. در گام بعد، تابع calc_cdf برای محاسبه cdf تصویر استفاده می‌شود. در انتها، تصویر جدید مطابق cdf محاسبه شده، توسط تابع reMap ساخته می‌شود. تابع calc_histogram به شکل زیر تعریف می‌شود:

```
def calc_hitogram(image):
    length = image.shape[0]
    width = image.shape[1]
    pdf = np.zeros(256)

    for i in range(length):
        for j in range(width):
            k = image[i][j]
            pdf[k] +=1

    return pdf
```

در این تابع، عکس مورد نظر به عنوان ورودی تابع استفاده می‌شود. در دو حلقه تو در تو، فراوانی رخ داد هر پیکسل که مقداری بین ۰ تا ۲۵۵ دارد را، در یک آرایه به نام pdf ذخیره می کنیم. طول این آرایه ۲۵۶ است، و اندیس k این آرایه، نشان دهنده فراوانی پیکسل هایی با مقدار k است.

برای اجرای روال متعادل سازی هیستوگرام، در ابتدا نیاز داریم تا طبق الگوریتم ارائه شده در درس، هیستوگرام فعلی را نرمالایز کنیم. برای اینکار یک تابع به نام normalizeHistogram نوشته ایم.

```
26  def normalizeHistogram(pdf,width,length):
27      normal_pdf = np.zeros(len(pdf))
28
29      for i in range(len(normal_pdf)):
30          normal_pdf[i] = pdf[i]/(width*length)
31
32      return normal_pdf
```

در این تابع، مقادیر pdf را به رنج بین ۰ تا ۱ مپ می‌کنیم. چون ماکسیمم مقدار فراوانی برابر حاصل ضرب طول در عرض تصویر است، برای تقسیم کردن در مخرج از این عدد استفاده می‌کنیم.

```
34  def calc_cdf(normal_pdf):
35      cdf = np.zeros(len(normal_pdf))
36
37      for i in range(len(normal_pdf)):
38          for j in range(i):
39              cdf[i] += normal_pdf[j]
40
41          cdf[i] *=255
42          cdf[i] = round(cdf[i])
43
44      return cdf
```

در این تابع، به ازای مقدا i ام از آرایه نرمال شده، تمام اندیس های ۰ تا i از آرایه نرمال شده را با هم جمع می‌کنیم. طبق فرمول محاسبه cdf، در حلقه بیرونی، که در واقع انتهای سیگمای جمع مقادیر نرمالایز شده تا آن اندیس است، مقدار آن اندیس را در ۲۵۵ ضرب می‌کنیم. در انتها چون این مقدار ممکن است یک عدد اعشاری باشد، آن را به مقدار صحیح نزدیک تر به آن گرد می‌کنیم.

```
46  def reMap(image,target_cdf):
47      newImage = image.copy()
48      for i in range(newImage.shape[0]):
49          for j in range(newImage.shape[1]):
50              newImage[i][j] = target_cdf[newImage[i][j]]
51
52      return newImage
```

این تابع با دریافت عکس هدف و cdf مورد نظر، پیکسل های عکس را با مقادیر جدید بر اساس cdf داده شده، آپدیت می‌کند.

در ادامه، ابتدا تصویر اصلی را با لول های مختلف، از ۲ تا ۱۲۸ لول، نمایش می‌دهیم.

```python
# quantize original image
img2level = quantize_simulation(elaine,1) # 2 level = 2**1
img4level = quantize_simulation(elaine,2) # 4 level = 2**2
img8level = quantize_simulation(elaine,3) # 8 level = 2**3
img16level = quantize_simulation(elaine,4) # 16 level = 2**4
img32level = quantize_simulation(elaine,5) # 32 level = 2**5
img64level = quantize_simulation(elaine,6) # 64 level = 2**6
img128level = quantize_simulation(elaine,7) # 128 level = 2**7
```

این قطعه کد، پیکسل های عکس elaine را با کمک تابع quantize_simulation ، در لول های مشخص شده، تقسیم بندی می کند.

```python
def quantize_simulation(image, n_bits):
    coeff = 2**8 // 2**n_bits
    return (image // coeff) * coeff
```

این تابع، با تقسیم۲۵۶ بر 2^{n_bits} ، تعداد دسته های مورد نظر را محاسبه می کند. سپس پیکسل های تصویر را بر این مقدار تقسیم و به پایین گرد می کند. در نهایت حاصل را دوباره در coeff ضرب می کنیم. منطق این عملیات این است که ابتدا پیکسل ها به اعدادی بین ۰ تا level-1 مقداری که میخواهیم نرمالایز می‌شوند، و در ادامه دوباره در coeff ضرب می کنیم تا به اعداد ابتدای بازه ها مپ شوند.

```python
# calculate MSE, for original image cases
mse2level = mean_square_error(elaine,img2level)
print(f'mean_square_error(2) = {mse2level}')
mse4level = mean_square_error(elaine,img4level)
print(f'mean_square_error(4) = {mse4level}')
mse8level = mean_square_error(elaine,img8level)
print(f'mean_square_error(8) = {mse8level}')
mse16level = mean_square_error(elaine,img16level)
print(f'mean_square_error(16) = {mse16level}')
mse32level = mean_square_error(elaine,img32level)
print(f'mean_square_error(32) = {mse32level}')
mse64level = mean_square_error(elaine,img64level)
print(f'mean_square_error(64) = {mse64level}')
mse128level = mean_square_error(elaine,img128level)
print(f'mean_square_error(128) = {mse128level}')
```

در نهایت در قطعه کد بالا، mean square error را بین عکس اصلی و عکس های گسسته سازی شده محاسبه می کند.

در سلول های بعد، همین روال را برای عکس متعادل شده تکرار کرده ایم.

نتیجه:

در ابتدا، با مقایسه عکس اصلی و متعادل شده، با توجه به عکس های حاصل، تاثیر متعادل سازی در افزایش کنتراست تصویرو جدا شدن لبه های سیاه از سفید و تشخیص بهتر جزئیات را مشاهده می‌کنیم.

در مورد گسسته سازی(quantize)، می‌بینیم که پس از ۸ یا ۱۶ سطح، دیگر تفاوت قابل تشخیص با چشم نیست.

در مورد محاسبه تفاوت بین عکس ها، اعداد زیر به‌دست آمد:

Level	***128***	***64***	***32***	***16***	***8***	***4***	***2***
Without Histeq	0.5093	3.4688	17.3819	77.3164	325.0481	1346.5181	5436.5977
With Histeq	0.5847	3.9539	17.8455	76.2318	325.0791	1318.1048	5296.0716

واضح است که هرچه تعداد سطوح به ۲۵۶ نزدیکتر باشد، مقدار خطا کمتر است. در مقایسه بین حالت متعادل شده و نشده نیز، تفاوت ارور برای عکس متعادل شده با عکس های گسسته اش، فقط در تعداد سطح پایین، خطای کمتری دارد، که البته تفاوت فاحشی نیست.

۱.۱.۲

توضیح فنی:

در ابتدا عکس را لود می کنیم.

```
goldhill = cv2.imread('Goldhill.bmp', cv2.IMREAD_GRAYSCALE)
width , length = goldhill.shape
plt.imshow(goldhill,cmap=plt.gray())
```

ابعاد عکس (512,512) است.

در حالت اول، عکس را با حذف سطر و ستون ها به صورت یکی در میان، کاهش ابعاد می‌دهیم.

```
# downsample an image by a factor of 2, without using averaging filter
# we just select rows & columns every other one
downsampled_without_averaging = goldhill[::2,::2]
plt.imshow(downsampled_without_averaging)
```

سپس، کاهش ابعاد را با استفاده از فیلتر میانگین گیر، انجام می‌دهیم.

```
# downsample an image by a factor of 2, using averaging filter
downsampled_using_averaging = averaging_filter(goldhill,windowSize=3,downsaplingFactor=2)
plt.imshow(downsampled_using_averaging)
```

در اینجا، از تابع averaging_filter استفاده شده است، که به شکل زیر پیاده‌سازی شده:

```python
def averaging_filter(image,windowSize=3,downsaplingFactor=1):
    #  apply averaging filter with 'windowSize' as filter size and then down-sampl
    newWidth = int(image.shape[0]/downsaplingFactor)
    newLength = int(image.shape[1]/downsaplingFactor)
    newImage = np.zeros((newWidth,newLength))

    for i in range(0, image.shape[0], downsaplingFactor):
        for j in range(0, image.shape[1], downsaplingFactor):
            end0 = i+windowSize
            if end0>image.shape[0]:
                end0 = image.shape[0]

            end1 = j+windowSize
            if end1>image.shape[1]:
                end1 = image.shape[1]

            buffer = np.mean(image[i:end0, j:end1], axis=(0,1))
            newImage[int(i/downsaplingFactor), int(j/downsaplingFactor)] = buffer

    return newImage
```

ورودی این تابع به ترتیب، عکس ورودی مورد نظر جهت کاهش، سایز پنجره فیلتر میانگین گیر و نرخ کاهش حجم است.

در ابتدا، یک آرایه جهت نگهداری عکس خروجی می‌سازیم. سپس در دو حلقه تو در تو، ابتدا، یک برش از عکس در ابعاد فیلتر را انتخاب می‌کنیم، سپس میانگین مقادیر پیکسل های این بخش را محاسبه می‌کنیم که در متغییر buffer ریخته می‌شود، و در انتها این مقدار را در پیکسل متناظر در آرایه عکس خروجی قرار می‌دهیم.

```python
# upsample an image by a factor of 2, using replication method
figure = plt.figure(figsize=(10,10))

upsampled_with_replication_1 = replication(downsampled_without_averaging,upsamplngFactor=2)
figure.add_subplot(1,2,1)
plt.imshow(upsampled_with_replication_1)
plt.title('upsampled removed rows & columns')

upsampled_with_replication_2 = replication(downsampled_using_averaging,2)
figure.add_subplot(1,2,2)
plt.imshow(upsampled_with_replication_2)
plt.title('upsampled averaged')
```

در قطعه کد بالا، با متد replication، ابعاد تصاویر را دو برابر می‌کنیم. پیاده‌سازی این متد به شکل زیر است:

```python
def replication(image,upsamplngFactor=2):
    # upsampling, by copying rows & columns
    newWidth = image.shape[0]*upsamplngFactor
    newLength = image.shape[1]*upsamplngFactor
    newImage = np.zeros((newWidth,image.shape[1]))

    for i in range(image.shape[0]):
        for k in range(upsamplngFactor):
            newImage[(i*upsamplngFactor)+k][:] = image[i][:]

    newImage2 = np.zeros((newWidth,newLength))

    for j in range(newImage.shape[1]):
        for k in range(upsamplngFactor):
            newImage2[: ,(j*upsamplngFactor)+k] = newImage[:,j]

    return newImage2
```

پارامتر های این تابع، تصویر ورودی برای تغییر ابعاد و نرخ تبدیل است. در این تابع، ابتدا ستون های عکس اصلی را در ستون های عکس خروجی، کپی و تکرار می‌کنیم، و سپس همین کار را برای سطرهای تصویر، تکرار می‌کنیم.

در ادامه اینبار با استفاده از متد دوخطی، افزایش ابعاد را انجام دادیم.

```python
# upsample an image by a factor of 2, using bilinear interpolation method
figure = plt.figure(figsize=(10,10))

upsampled_with_bilinear_interpolation_1 = np.zeros((downsampled_without_averaging.shape[0]*2,downsampled_without_
upsampled_with_bilinear_interpolation1 = interpolate_bilinear(downsampled_without_averaging,upsampled_with_biline
figure.add_subplot(1,2,1)
plt.imshow(upsampled_with_bilinear_interpolation_1)
plt.title('bilinear_interpolation_1')

upsampled_with_bilinear_interpolation_2 = np.zeros((downsampled_using_averaging.shape[0]*2,downsampled_using_aver
upsampled_with_bilinear_interpolation2 = interpolate_bilinear(downsampled_using_averaging,upsampled_with_bilinear
figure.add_subplot(1,2,2)
plt.imshow(upsampled_with_bilinear_interpolation_2)
plt.title('bilinear_interpolation_2')
```

پیاده‌سازی متد interpolate_bilinear با کمک یک منبع خارجی، به شکل زیر است.

```
104  def interpolate_bilinear(array_in, array_out):
105      width_in = array_in.shape[0]
106      height_in = array_in.shape[1]
107      width_out = array_out.shape[0]
108      height_out = array_out.shape[1]
109
110      for i in range(height_out):
111          for j in range(width_out):
112              # Relative coordinates of the pixel in output space
113              x_out = j / width_out
114              y_out = i / height_out
```

ورودی این تابع به ترتیب، عکس ورودی و آرایه با ابعاد مدنظر برای عکس خروجی است.

```
116              # Corresponding absolute coordinates of the pixel in input space
117              x_in = (x_out * width_in)
118              y_in = (y_out * height_in)
119
120              # Nearest neighbours coordinates in input space
121              x_prev = int(np.floor(x_in))
122              x_next = x_prev + 1
123              y_prev = int(np.floor(y_in))
124              y_next = y_prev + 1
```

```
126                # Sanitize bounds - no need to check for < 0
127                x_prev = min(x_prev, width_in - 1)
128                x_next = min(x_next, width_in - 1)
129                y_prev = min(y_prev, height_in - 1)
130                y_next = min(y_next, height_in - 1)
131
132                # Distances between neighbour nodes in input space
133                Dy_next = y_next - y_in;
134                Dy_prev = 1. - Dy_next; # because next - prev = 1
135                Dx_next = x_next - x_in;
136                Dx_prev = 1. - Dx_next; # because next - prev = 1
```

```
138            # Interpolate over 3 RGB layers
139            if (len(array_out.shape) > 2):
140                for c in range(3):
141                    array_out[i][j][c] = Dy_prev * (array_in[y_next][x_prev][c] * Dx_next + array_in[y_next
142                    + Dy_next * (array_in[y_prev][x_prev][c] * Dx_next + array_in[y_prev][x_next][c] * Dx_pr
143            else: # Interpolate over 1 grayscale layer
144                array_out[i,j] = Dy_prev * (array_in[y_next][x_prev] * Dx_next + array_in[y_next][x_next] *
145            + Dy_next * (array_in[y_prev][x_prev] * Dx_next + array_in[y_prev][x_next] * Dx_prev)
146
147    return array_out
```

در هر بخش از کد، کامنت ها، گویای عملیات ساده ای است که رخ می‌دهد، پس توضیح اضافه ای ندارد.

در انتها، تفاوت عکس ها را با تصویر اصلی محاسبه کرده‌ایم.

```python
# calculate MSE
removal_down_replication_up = mean_square_error(goldhill,upsampled_with_replication_1)
print(f'(MSE) removal_down_replication_up: {removal_down_replication_up}')

removal_down_bilinear_up = mean_square_error(goldhill,upsampled_with_bilinear_interpolation1)
print(f'(MSE) removal_down_bilinear_up: {removal_down_bilinear_up}')

averaging_down_replication_up = mean_square_error(goldhill,upsampled_with_replication_2)
print(f'(MSE) averaging_down_replication_up: {averaging_down_replication_up}')

averaging_down_bilinear_up = mean_square_error(goldhill,upsampled_with_bilinear_interpolation2)
print(f'(MSE) averaging_down_bilinear_up: {averaging_down_bilinear_up}')
```

نتیجه:

Pixel Replication	*Bilinear Interpolation*	
99.0850	142.4705	*Averaging*
133.0759	65.0079	*Remove Row&Column*

با توجه به مقادیر خطا محاسبه شده، بهترین روش برای کاهش ابعاد و سپس افزایش مجدد ابعاد یک تصویر، استفاده از روش حذف سطر و ستون و در ادامه تبدیل دوخطی است.

۱.۱.۳

توضیح فنی:

```python
barbara = cv2.imread('Barbara.bmp', cv2.IMREAD_GRAYSCALE)

# plt.imshow(barbara,cmap=plt.gray())
img8bit = get_bit_planes(barbara.copy(), 0b11111111)
img7bit = get_bit_planes(barbara.copy(), 0b01111111)
img6bit = get_bit_planes(barbara.copy(), 0b00111111)
img5bit = get_bit_planes(barbara.copy(), 0b00011111)
img4bit = get_bit_planes(barbara.copy(), 0b00001111)
img3bit = get_bit_planes(barbara.copy(), 0b00000111)
img2bit = get_bit_planes(barbara.copy(), 0b00000011)
img1bit = get_bit_planes(barbara.copy(), 0b00000001)

```

پس از لود تصویر، با استفاده از تابع، get_bit_planes ، تصاویر را با تعداد بیت های به ترتیب ۸ تا ۱ می‌سازیم.

پیاده‌سازی تابع get_bit_planes به شکل زیر است:

```python
def get_bit_planes(image, bit_planes):
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            image[i][j] = image[i][j] & bit_planes
    image = (((image - image.min()) / (image.max() - image.min())) * 255.0).astype('uint8')
    return image

```

در این تابع، رشته بیتی مدنظر را با مقدار پیکسل های تصویر، اند منطقی می گیریم تا فقط اطلاعات بیت های مدنظر، باقی بماند.

نتیجه:

با توجه به نتیجه عکس های خروجی، می‌توان گفت، با ۴ یا ۵ بیت، دیگر نمی توان اطلاعات خوبی از تصویر داشت. البته در پاسخ به سوال دوم مطرح شده، می‌توان گفت در بخش هایی که فرکانس تغییرات پایین تر است، یا به بیانی، با بخش های یکدست تصویر رو به رو هستیم، می توان تصویر را با تعداد بیت های کمتر، و با کیفیت قبل نشان داد، اما بخش هایی که فرکانس تغییرات زیادتر است، به سرعت تصویر نویزی و خراب می‌شود(حالت برفکی پیدا می کند)